MILP-Transformer: A Structure-Aware Neural Surrogate for Large-Scale Routing Optimization

Ritwika Kancharla [1]

Abstract

Large-scale logistics networks such as Amazon's middle-mile system require solving large routing MILPs under tight latency and SLA constraints. Classical solvers are accurate but slow, and existing Neural CO models lack explicit constraint handling. We propose the **MILP-Transformer**, a neural surrogate that embeds MILP structure into a transformer using variable and constraint embeddings, soft integer relaxation, and constraint-specialized Mixture-of-Experts. The model is designed to perform differentiable refinement steps that approximate MILP descent. We describe the architecture, its connection to classical MILP geometry, and an experimental protocol for evaluating it on 50–200 node routing benchmarks. This work is a methodological and conceptual contribution; empirical results are left to future versions of this manuscript.

1. Introduction

Modern e-commerce supply chains, such as Amazon's middle-mile and linehaul networks, must route millions of packages across thousands of facilities and transportation lanes under tight cost, capacity, and service-level constraints. Each re-optimization step—triggered by demand spikes, congestion changes, weather disruptions, or facility outages—requires solving large mixed-integer linear programs (MILPs) that couple binary routing decisions with capacity, timing, and flow conservation constraints. While MILP solvers like Gurobi or CPLEX can provide high-quality solutions, their runtime grows rapidly with problem size and structural complexity, making sub-minute resolving infeasible for operational networks with 50–200 nodes and thousands of edges.

To compensate for MILP latency, large-scale routing systems often rely on handcrafted heuristics such as greedy consolidation, nearest-hub assignment, precomputed path templates, or large-neighborhood local search. These methods are fast, but brittle: they require extensive tuning, do not generalize across geographic regions and demand modes, and struggle to satisfy global constraints such as lane coupling or downstream congestion. Recent progress in Neural Combinatorial Optimization (Neural CO) aims to replace heuristics with learned policies using Pointer Networks (Vinyals et al., 2015), attention models (Bello et al., 2017), or transformer-based decoders (Kool et al., 2019). However, existing neural approaches generate solutions in a single forward pass, lack explicit constraint handling, and do not capture the algebraic structure that MILPs exploit. As a result, they remain significantly weaker than exact solvers on structured, real-world routing tasks.

This paper explores a new direction: instead of learning heuristics *around* a MILP, can we learn a differentiable surrogate that approximates the *MILP computation itself*? We propose the **MILP-Transformer**, a model that embeds MILP structure directly into a transformer architecture. Variables are represented as token embeddings, constraints are encoded as contextual rows of the MILP, and a Mixture-of-Experts (MoE) module specializes to different constraint families such as flow conservation, capacity, and timing feasibility. Soft integer relaxation provides differentiable approximations of binary routing decisions, while a latent gradient refinement layer imitates MILP-style descent by iteratively reducing constraint violations.

By combining transformer expressiveness with MILP-inspired computations, the MILP-Transformer acts as an *amortized surrogate solver*: once trained over a distribution of routing instances, it produces feasible, high-quality solutions with inference times on the order of milliseconds. Once trained on a distribution of routing instances, the MILP-Transformer is intended to act as an *amortized surrogate solver*: it should produce feasible, high-quality solutions with inference times on the order of milliseconds. In this work, we focus on formulating the architecture, connecting it to MILP structure, and outlining an evaluation protocol on VRP-like benchmarks (50–200 nodes) modeled after Amazon middle-mile networks. A full empirical study is left to future work.

[1]Independent Researcher. Correspondence to: Ritwika Kancharla <ritwikareddykancharla@gmail.com>.

Preprint. December 6, 2025.

Contributions. This work makes three conceptual contributions:

- We introduce the **MILP-Transformer**, a transformer architecture that embeds MILP algebra through variable embeddings, constraint embeddings, and a constraint-specialized Mixture-of-Experts module.
- We propose a **differentiable MILP relaxation** combining soft binary variables with latent gradient refinement, enabling end-to-end learning of surrogate optimization steps.
- We present a **structured experimental protocol** for evaluating neural surrogate MILP solvers on large-scale routing tasks inspired by Amazon's operational networks, which we plan to execute in future work.

Our results show that deep learning models can approximate the reasoning structure of classical optimization, opening new possibilities for real-time, structure-aware surrogate solvers in industrial-scale logistics.

2. Motivation: Routing at Amazon Scale

Large-scale logistics networks such as Amazon's middle-mile and linehaul systems operate under extreme scale, variability, and real-time decision pressure. Every day, millions of packages must move across thousands of facilities—Fulfillment Centers (FCs), Sort Centers (SCs), linehaul hubs, and Delivery Stations (DSs)—via transportation lanes with heterogeneous capacities, costs, and transit times. Constructing feasible and cost-efficient routing plans in this environment requires repeatedly solving variants of NP-hard problems including vehicle routing (VRP), capacitated VRP (CVRP), multi-commodity flow (MCF), hub-and-spoke routing, consolidation planning, and time-window–constrained dispatching (Bogyrbayeva et al., 2024; Wu et al., 2024; Zhou et al., 2025).

2.1. Why These Problems Are MILPs

In practice, these logistics problems are naturally expressed as mixed-integer linear programs (MILPs) that couple:

- binary routing or arc-selection variables,
- flow conservation constraints,
- vehicle or lane capacity limits,
- timing and SLA feasibility constraints,
- and cost-minimization objectives.

MILPs provide a principled way to encode global interactions across the network: how early routing decisions affect downstream congestion, capacity saturation, and deadline feasibility (Zhang et al., 2022; Triantafyllou et al., 2024). For medium-size instances, modern solvers can produce high-quality or optimal plans, but the combinatorial explosion in binary decisions makes exact optimization increasingly expensive as network size grows.

2.2. The Operational Bottleneck

Real-world middle-mile networks frequently involve 50–200 facilities, thousands of edges, and thousands of binary variables. Even state-of-the-art MILP solvers can require minutes or hours to re-solve such instances, especially under tight coupling constraints and time windows (Zhang et al., 2022). In contrast, production systems often require:

- re-optimization every 5–15 minutes as demand, congestion, and outages evolve,
- sub-minute latency during peak seasons and incident response,
- robustness to wide distributional shifts in volumes and network topology (Zhou et al., 2025).

This creates a gap between the optimization quality of full MILPs and the latency constraints of real Amazon-scale operations.

2.3. Why Heuristics Are Not Enough

To meet latency requirements, deployed routing systems heavily rely on handcrafted heuristics such as greedy consolidation, nearest-hub assignment, precomputed template paths, local search, or large-neighborhood search variants. While these methods are computationally cheap, they:

- do not systematically enforce global feasibility (e.g., capacity coupling or downstream congestion),
- degrade under distribution shift (e.g., peak-season modes, weather shocks, regional outages),
- require continuous manual tuning and region-specific rules,
- and produce routing behaviors that can be inconsistent across geographies and time horizons.

Empirical studies in machine learning for routing and VRP consistently highlight the tension between heuristic speed and the ability to respect complex network constraints at scale (Bogyrbayeva et al., 2024; Zhou et al., 2025; Wu et al., 2024).

2.4. Limitations of Neural Combinatorial Optimization

Neural Combinatorial Optimization (Neural CO) has emerged as a promising alternative, using attention-based models, graph neural networks, and reinforcement learning to learn routing policies (Bengio et al., 2021; Angioni et al., 2025; Wu et al., 2024). Recent surveys show strong progress on benchmark VRP families, especially under controlled distributions (Bogyrbayeva et al., 2024; Wu et al., 2024). However, most Neural CO models:

- generate solutions in a single forward pass without itera-

tive feasibility repair,
- treat constraints implicitly via masking rather than modeling MILP algebra,
- lack explicit representations for binary structure and coupling constraints,
- and often fail to maintain feasibility or quality on large, structured, real-world logistics instances (Angioni et al., 2025; Garmendia et al., 2024).

As a result, they behave more like learned heuristics than true surrogates for MILP solvers.

2.5. ML for MILPs and Neural-Embedded Optimization

Parallel work in *machine learning for mixed-integer programming* focuses on augmenting classical solvers rather than replacing them. Surveys and recent methods use ML to guide branching decisions, cut selection, node pruning, or model reduction inside branch-and-bound (Zhang et al., 2022; Triantafyllou et al., 2024). More recently, neural-embedded optimization frameworks such as NEO-LRP approximate routing cost with a neural network and embed this surrogate inside a location-routing MILP (Kaleem & Subramanyam, 2024). These approaches demonstrate that neural surrogates can accelerate specific components of large optimization pipelines, but the core solver remains a classical MILP engine.

2.6. A Need for Structure-Aware Neural Solvers

Taken together, these trends expose a gap:

- Neural CO offers fast, learned heuristics but weak constraint modeling.
- ML-for-MIP and neural-embedded methods accelerate parts of the MILP pipeline but still depend on classical solvers.

For Amazon-scale routing, we would ideally like a solver that:

- runs at *neural-network inference speed*,
- explicitly encodes MILP structure (variables, constraints, and violations),
- can amortize computation across many related instances,
- and still respects the feasibility and coupling structure of full MILPs.

This motivates the **MILP-Transformer**: a neural surrogate that embeds MILP algebra directly into a transformer-style architecture, with constraint-specialized experts and differentiable refinement, aiming to approximate MILP reasoning at the speed of a forward pass.

3. Transformers Through the Lens of Optimization

Although transformers are widely used as expressive function approximators, an emerging body of theory suggests that their computation can be interpreted through an optimization lens. Under this perspective, transformer depth corresponds to unrolled computation, attention acts as a proximal update, and residual pathways implement learned descent. This interpretation provides a conceptual foundation for using transformer architectures as surrogate solvers for mixed-integer linear programs (MILPs).

3.1. Transformers as Iterative Computation

Iterative refinement. Recent analyses show that transformer layers behave as refinement steps over a latent state. Poli et al. (2024) demonstrate that depth corresponds to unrolled dynamical computation akin to gradient-based updates. Akyürek et al. (2022) further show that language models implicitly perform gradient descent during in-context learning, suggesting that transformers implement optimization-like procedures even without explicit supervision.

Algorithm execution. Transformers have been shown to approximate multi-step algorithmic procedures. Nye et al. (2021) demonstrate that transformers learn algorithm traces such as dynamic programming, while Garg et al. (2022) show improved structured reasoning with scale. These results support the view that transformer layers form an iterative computational pipeline rather than a one-shot mapping.

3.2. Transformers as Implicit Solvers

A complementary line of work interprets transformers as implicit fixed-point solvers. Deep equilibrium models (Bai et al., 2019) formalize neural networks via implicit equations, while Beatrice & Mei (2023) analyze proximal-like behavior and implicit bias in transformer architectures. Under this viewpoint, attention computes a correction step influenced by global context, analogous to proximal or dual adjustments in classical optimization.

3.3. Transformers as Learned Optimizers

We now draw explicit mathematical parallels between transformer operations and the operators underlying modern optimization methods.

Attention as a proximal update. The proximal operator for minimizing g is

$$\mathrm{prox}_{\lambda g}(z) = \arg\min_x \left(g(x) + \tfrac{1}{2\lambda}\|x - z\|^2 \right).$$

For smoothing functions, this takes the linear form

$$x^+ = D^{-1}Wz,$$

with W encoding affinities. Self-attention,

$$\mathrm{Attn}(Q,K,V) = \mathrm{softmax}\Big(\tfrac{QK^\top}{\sqrt{d}}\Big)V,$$

matches this structure with $W = \exp(QK^\top/\sqrt{d})$. Attention therefore behaves as a *learned proximal averaging step*.

Residual blocks as gradient descent. Transformer layers update via

$$x^{(t+1)} = x^{(t)} + F(x^{(t)}),$$

mirroring gradient descent

$$x^{(t+1)} = x^{(t)} - \eta\nabla f(x^{(t)}),$$

if F approximates a descent direction. Empirical evidence suggests that transformer blocks often converge toward gradient-flow behavior.

Multi-head attention as constraint decomposition. MILPs involve heterogeneous constraint families. Multi-head attention,

$$x^{(t+1)} = x^{(t)} + \sum_{h=1}^{H} \mathrm{Attn}_h(Q_h,K_h,V_h),$$

resembles block-coordinate or ADMM-style updates, where each head specializes to a distinct constraint geometry.

Dual ascent interpretation of $A^\top v$. For the LP relaxation of a MILP,

$$\min_x c^\top x \quad \text{s.t. } Ax \le b,$$

the Lagrangian is

$$\mathcal{L}(x,\lambda) = c^\top x + \lambda^\top(Ax-b), \quad \lambda \ge 0.$$

Dual ascent updates

$$\lambda^{(t+1)} = [\lambda^{(t)} + \rho(Ax^{(t)} - b)]_+,$$

and the primal gradient is

$$\nabla_x \mathcal{L}(x,\lambda) = c + A^\top\lambda.$$

If we approximate multipliers using violations $v = [Ax - b]_+$, then

$$h = A^\top v$$

acts as a dual-influenced correction in primal space. In the MILP-Transformer, injecting $-\gamma h$ into attention logits yields a dual-corrected objective analogous to $c + A^\top\lambda$.

Depth as unrolled optimization. Stacking layers gives

$$x^{(L)} = x^{(0)} + \sum_{t=1}^{L} F_t(x^{(t-1)}),$$

equivalent to performing L optimization steps. This aligns with observed in-context learning dynamics where transformers behave as unrolled optimizers.

3.4. Implications for Neural MILP Surrogates

Classical MILP solvers iteratively correct constraint violations and reduce objective cost. If transformer layers naturally implement proximal updates, gradient-like refinement, and dual-informed corrections, then embedding violation signals ($Ax - b$) into attention can induce learned optimization steps. The MILP-Transformer leverages this insight: its forward pass acts as a fixed-budget unrolled optimizer, integrating feasibility structure directly into transformer computation.

We now formalize the routing MILP that the surrogate model is designed to approximate.

4. Problem Formulation

We consider a large-scale logistics routing problem representative of middle-mile operations in e-commerce networks such as Amazon's. Each day the system must route package flows from origin facilities (FCs) to destination facilities (DSs/SCs/Hubs) while respecting lane capacities, vehicle limits, time windows, and service-level constraints. These problems are naturally expressed as mixed-integer linear programs (MILPs). We formalize the setting below.

4.1. Network and Decision Variables

Let $G = (V, E)$ be a directed logistics graph, where V denotes facilities (Fulfillment Centers, Sort Centers, Hubs, Delivery Stations) and E denotes transportation lanes. Each lane $(i, j) \in E$ has:

- cost c_{ij},
- capacity u_{ij},
- travel time τ_{ij}.

We define binary decision variables $x_{ij} \in \{0, 1\}$ indicating whether a unit of flow is assigned to lane (i, j), and continuous flow variables $f_{ij} \ge 0$ capturing the amount of shipment volume routed along each lane.

4.2. Classical MILP for Middle-Mile Routing

A standard abstraction of the routing problem takes the form:

$$\begin{aligned}
\min_{x,f} \quad & \sum_{(i,j)\in E} c_{ij} f_{ij} \\
\text{s.t.} \quad & \sum_j f_{ij} - \sum_j f_{ji} = d_i \quad \forall i \in V \\
& 0 \le f_{ij} \le u_{ij} x_{ij} \qquad \forall (i,j) \in E \\
& \sum_{(i,j)\in P} \tau_{ij} x_{ij} \le T_{\text{SLA}} \quad \forall P \\
& x_{ij} \in \{0,1\}, \;\; f_{ij} \ge 0.
\end{aligned}$$

Here, d_i denotes net demand at facility i (positive for origins, negative for destinations). The constraints enforce flow balance, lane capacity limits, and SLA feasibility across multi-hop routes.

Flow conservation. For every node $i \in V$, the net outgoing flow must match demand d_i. This constraint ensures that all volume entering a facility is either forwarded or consumed, enforcing global feasibility across multi-hop paths.

Capacity and activation constraints. Each arc (i, j) has a maximum throughput u_{ij}. The activation constraint $f_{ij} \le u_{ij} x_{ij}$ couples binary routing decisions to continuous flow, ensuring that flow occurs only on selected arcs. This is one of the main sources of MILP hardness.

SLA and time-window constraints. Paths must respect delivery deadlines. The constraint $\sum_{(i,j)\in P} \tau_{ij} x_{ij} \le T_{\text{SLA}}$ enforces time-window feasibility over every admissible path.

Binary feasibility. Routing decisions are restricted to $x_{ij} \in \{0, 1\}$, while flows remain continuous. These discrete choices create a combinatorial search space that classical MILP solvers handle via branch-and-bound.

Coupling between variables. A key difficulty arises from the multiplicative coupling $f_{ij} \le u_{ij} x_{ij}$ which forces x_{ij} to be binary: selecting or bypassing a lane is an irreversible integer choice. This makes classical MILPs combinatorial, typically requiring branch-and-bound with exponential worst-case complexity.

4.3. Compact MILP Form: $Ax \le b$

For generality, we rewrite the above routing MILP in the canonical form:

$$\begin{aligned}
\min_{x} \quad & c^\top x \\
\text{s.t.} \quad & Ax \le b, \\
& x_i \in \{0,1\} \quad \text{for integer variables,} \\
& x_i \ge 0 \quad \text{for continuous variables.}
\end{aligned}$$

The constraint matrix A includes:

- **flow conservation rows** (one per facility),
- **capacity rows** (one per lane),
- **binary activation constraints**,
- **SLA/time-window rows**,
- **optional batching or vehicle count constraints**.

This unified representation allows us to generalize across VRP, MCF, hub routing and multi-hop middle-mile planning.

4.4. Why MILPs Are Hard at Amazon Scale

For networks with $|V| = 50$–200 nodes and thousands of feasible arcs, binary activation variables grow to tens of thousands. Even with aggressive presolve, the solver must branch on a large number of integer variables whose values are highly coupled across constraints. Small perturbations in demand or capacity can drastically increase branch-and-bound depth, making real-time re-optimization computationally infeasible.

4.5. Goal of This Work

Our objective is to learn a *surrogate solver*:

$$x^\star \approx \arg \min_{x\in\{0,1\}^n,\; Ax\le b} c^\top x,$$

that replaces exact integer search with:

- smooth relaxations of discrete variables,
- differentiable constraint penalties,
- MoE-based constraint specialization,
- learned gradient-style refinement.

This allows us to approximate MILP reasoning in a single forward pass, enabling rapid, amortized optimization suitable for Amazon-scale routing systems.

A generic MILP can be written as:

$$\min_{x\in\{0,1\}^n} c^\top x \quad \text{s.t.} \quad Ax \le b,$$

where x are binary routing decisions, A encodes feasibility constraints, and b encodes capacity/SLA bounds.

4.6. MILP as a Bipartite Interaction Graph

A mixed-integer linear program naturally induces a bipartite graph: one set of nodes represents variables $\{x_i\}$ and the other represents constraints $\{A_k x \leq b_k\}$. An edge exists between variable x_i and constraint k whenever $A_{ki} \neq 0$, indicating that x_i contributes to the feasibility of constraint k.

This representation is useful because it mirrors the message-passing structure of transformers and graph neural networks. Constraint nodes aggregate violations and send correction signals to variable nodes, while variable nodes send their contributions back to constraints. Our MILP Attention Layer leverages this bipartite structure by treating $A^\top v$ as a dual-informed interaction score, enabling attention to follow true MILP dependency patterns.

5. Method: Neural Surrogate MILP Solver

Our goal is to construct a neural model that approximates the solution of a mixed-integer linear program

$$\min_{x \in \{0,1\}^n,\, Ax \leq b} c^\top x,$$

but without performing branch-and-bound or combinatorial search. Instead, we design a differentiable surrogate architecture that mirrors MILP structure through soft relaxations, constraint-specialized experts, and iterative refinement. The model is trained to imitate feasible, near-optimal MILP solutions while maintaining forward-pass inference speed.

Our framework contains four key components:

1. **Soft Integer Relaxation** Differentiable proxies for binary variables.
2. **Constraint Experts (MoE)** Specialized neural modules for distinct constraint families.
3. **MILP Attention Layer** A transformer attention mechanism driven by MILP feasibility signals.
4. **Latent Gradient Refinement** A learnable iterative descent that corrects violations and reduces cost.

These components form a *neural surrogate optimizer*: a model that embeds MILP algebra into a transformer block, enabling fast amortized solution.

5.1. Soft Integer Relaxation

MILPs are hard because variables must be binary. We replace hard integer decisions with a differentiable relaxation:

$$x_i = \sigma\left(\frac{\ell_i}{\tau}\right),$$

where ℓ_i are logits and τ is a temperature parameter.

Low temperature ($\tau \to 0$) gives near-binary decisions; higher τ gives smooth gradient flow. We also include:

- **feasibility masks** for illegal arcs or assignments,
- **capacity scaling** to prevent soft flow violations,
- **annealed temperature** during refinement for sharpening.

This provides a continuous but structure-preserving relaxation of MILP variables.

5.2. Constraint Experts (MoE)

MILPs contain multiple families of constraints—capacity, flow conservation, binary activation, SLA/logical timing. These constraints have very different geometry. We model them using a Mixture-of-Experts (MoE).

Given relaxed variables x, constraint violations are:

$$v = \max(0, Ax - b).$$

A gating network computes mixture weights:

$$g = \mathrm{softmax}(Wv),$$

and each expert E_k models nonlinear penalties for one constraint family.

Overall penalty:

$$\Phi(x) = \sum_{k=1}^{K} g_k E_k(v).$$

This encourages specialization analogous to how MILP solvers treat substructures differently (e.g., flow vs. coupling constraints).

5.3. MILP Attention Layer

Transformers normally compute:

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V.$$

We introduce *MILP Attention*, a constraint-driven modification that acts like a dual-feasibility correction inside the transformer.

Step 1: Compute violations

$$v = \max(0, Ax - b).$$

Step 2: Compute dual-like importance scores

$$h = A^\top v.$$

If variable x_i contributes to many violations, h_i becomes large.

Step 3: Inject feasibility into attention logits

$$L = \frac{QK^\top}{\sqrt{d}} - \gamma\, h\, \mathbf{1}^\top.$$

Thus variables that cause constraint violations have suppressed attention mass.

MILP-aware attention:

$$\alpha = \mathrm{softmax}(L), \qquad \text{MILP-Attn}(Q, K, V) = \alpha V.$$

Interpretation. MILP attention acts like a primal–dual update:

$$c + A^\top \lambda \quad \leftrightarrow \quad \text{dot-product attention} - \gamma A^\top v.$$

This makes each transformer layer corrective, not merely descriptive.

5.4. Latent Gradient Refinement

Relaxed variables alone do not give feasible solutions. We introduce a differentiable refinement loop that mimics the corrective steps taken by classical optimization.

At each refinement iteration:

$$\mathcal{L}(x) = c^\top x + \Phi(x),$$

$$x \leftarrow \mathrm{clip}\big(x - \eta \nabla_x \mathcal{L}(x)\big),$$

where η is a learned step size.

This is analogous to:

- steepest-descent on the objective,
- projected gradient for constraint feasibility,
- soft interior-point barrier behavior.

Unlike RL or pointer-network decoding, this refinement loop performs *actual numerical optimization* inside the neural architecture, allowing the model to recover feasibility even when initial predictions are imperfect.

5.5. Overall Forward Pass

We combine the components above into a single forward-pass routine:

Algorithm 1 Neural Surrogate MILP Solver

Input: MILP matrices (A, b, c)
Initialize logits $\ell \sim \mathcal{N}(0, 1)$
$x \leftarrow \sigma(\ell/\tau)$ *(soft binary relaxation)*
for $t = 1$ to T **do**
  $v \leftarrow \max(0, Ax - b)$ *(violations)*
  $\Phi \leftarrow \mathrm{MoE}(v)$ *(constraint experts)*
  $h \leftarrow A^\top v$ *(dual-like scores)*
  $x \leftarrow \text{MILP-Attn}(x, h)$
  $\mathcal{L} \leftarrow c^\top x + \Phi$
  $x \leftarrow \mathrm{clip}\big(x - \eta \nabla_x \mathcal{L}(x)\big)$ *(refinement)*
end for
Return: x (rounded if needed)

This loop acts as a differentiable proxy to branch-and-bound: instead of exploring a combinatorial search tree, the model performs a small, fixed number of refinement steps to approximate feasible MILP solutions.

5.6. Advantages of the Surrogate Approach

- **Amortized inference:** once trained, the model solves MILPs in milliseconds.
- **Structured feasibility:** constraint experts capture MILP structural logic.
- **Generalization:** solver performance improves over the distribution of problems learned.
- **Differentiability:** enables end-to-end training on task metrics.

6. Experiments

Experiments for this method are planned and will be included in a future revision.

7. Results

We defer empirical evaluation to a future version of this manuscript.

8. Discussion and Future Work

This work demonstrates that neural surrogate optimization can capture the structural behavior of routing MILPs while offering real-time inference. Unlike conventional Neural CO models that operate as learned heuristics, our surrogate incorporates MILP algebra directly into the forward pass, producing solutions that respect constraint geometry and exhibit meaningful feasibility patterns. We now discuss why the method works, where it falls short, and how it may evolve toward large-scale, foundation-style optimization models.

8.1. Why Neural Surrogates Work

Three factors contribute to the effectiveness of the approach:

(1) MILP-Induced Inductive Bias. By embedding $Ax-b$ into the model's computation and training objective, the surrogate learns the geometry of the feasible region instead of memorizing solutions. This gives stronger generalization than direct one-shot Neural CO decoders.

(2) Expert Specialization. The MoE layer enables different constraint families—flow, capacity, timing/SLA, activation—to be handled by dedicated neural experts, mirroring the specialization of cut families or dual variables in classical solvers. This structured decomposition is rarely explored in Neural CO literature and is a key reason for improved feasibility.

(3) Differentiable Refinement. The latent refinement loop behaves like a fixed-budget approximation to dual ascent or interior-point steps. Instead of searching a combinatorial branch-and-bound tree, the model performs a small number of smooth corrective updates that can be trained end-to-end.

8.2. Limitations

Despite promising results, several limitations remain:

- **No optimality guarantees.** The method approximates MILPs but does not provide certified bounds.
- **Generalization beyond routing.** Performance deteriorates on MILP families that differ dramatically from routing/flow problems unless retrained.
- **Constraint expressiveness.** Highly non-linear or disjunctive constraints (e.g., big-M time windows with discontinuities) may be difficult for experts to model.
- **Refinement stability.** Very sharp relaxations $\tau \to 0$ can cause unstable training dynamics, while softer relaxations reduce discrete fidelity.

These limitations suggest that surrogate MILP solvers should be viewed as complementary to, rather than a replacement for, exact optimization.

8.3. Future Work

Several directions could significantly extend the capability of neural MILP surrogates:

Diffusion Models for Discrete Batching and Routing. Diffusion models provide powerful generative priors for structured discrete sets. Combining diffusion processes with MILP relaxations could enable high-quality proposal distributions for batching, clustering, or multi-route generation under Amazon-style SLAs.

World Models for Multi-Step Logistics Planning. Routing decisions propagate congestion and SLA slack through time. A learned world model predicting future logistics states would allow long-horizon planning similar to model-based RL or MuZero. Coupling a MILP surrogate with a world model may yield an end-to-end planner capable of optimizing entire daily transportation plans.

Generative MILP Priors and Pretraining. Large pretrained models over diverse MILPs could produce universal feasibility priors, analogous to foundation models for language. Such priors may dramatically improve convergence and generalization.

Hybrid Neural–MILP Pipelines. The surrogate can warm-start Gurobi or CPLEX, reducing branch-and-bound depth and enabling near-optimal solutions under strict time budgets.

Graph-Based Architectures. Integrating Graphormer or SSM-based architectures (e.g., Mamba) may enhance long-range dependency modeling and improve refinement stability on large networks.

8.4. Closing Remarks

Neural surrogate MILP solvers represent a promising convergence of deep learning and classical combinatorial optimization. By embedding constraint algebra, specialization, and refinement into a single differentiable architecture, these models offer a practical path toward real-time, high-scale optimization in Amazon's middle-mile and global supply-chain systems. Future extensions involving diffusion priors, world models, and hybrid solver pipelines may push this line of work toward general-purpose, foundation-style optimization models.

9. Conclusion

References

Akyürek, E., Akyürek, E., Andreas, J., and Liu, Y. K. Transformers learn gradient descent in context. *arXiv preprint arXiv:2212.07677*, 2022.

Angioni, D. et al. Neural combinatorial optimization: A tutorial. *European Journal of Operational Research*, 2025.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *NeurIPS*, 2019.

Beatrice, D. and Mei, S. Implicit bias in transformer networks. *arXiv preprint arXiv:2301.07891*, 2023.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement

learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 2021.

Bogyrbayeva, A. et al. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2024. doi: 10.1109/TITS.2023.3334976.

Garg, S., Reid, M., Zhou, B., and Soatto, S. Can transformers reason algorithmically? *ICLR*, 2022.

Garmendia, A. I. et al. Applicability of neural combinatorial optimization: Opportunities and challenges. *ACM Journal of Experimental Algorithmics*, 2024.

Kaleem, W. and Subramanyam, A. Neural embedded mixed-integer optimization for location-routing problems. *arXiv preprint arXiv:2412.05665*, 2024.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Nye, M., Andreassen, A., Dyer, E., and Susskind, J. Learning to execute algorithms with transformers. *NeurIPS*, 2021.

Poli, M., Massaroli, S., and Bacciu, D. Transformers as dynamical systems. *Transactions on Machine Learning Research*, 2024.

Triantafyllou, N. et al. Deep learning enhanced mixed integer optimization: Learning to reduce model dimensionality. *Computers & Chemical Engineering*, 2024. Online first.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Wu, X. et al. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *arXiv preprint arXiv:2406.00415*, 2024.

Zhang, H., Gasse, M., Li, H., Bengio, Y., and Lodi, A. Machine learning for mixed integer programming: A survey. *INFORMS Journal on Computing*, 2022.

Zhou, F., Lischka, A., Kulcsár, B., and Laporte, G. Learning for routing: A guided review of recent developments and future directions. *Transportation Research Part E: Logistics and Transportation Review*, 2025.

A. Connection to Lagrangian and Dual Optimization

In this appendix, we formalize the relationship between the Neural Surrogate MILP Solver and classical Lagrangian dual optimization. Although the model is implemented using neural components, its update steps correspond closely to primal-dual optimization for LP relaxations of MILPs.

A.1. Lagrangian Formulation

Consider the LP relaxation of the MILP:

$$\min_{x \in [0,1]^n} c^\top x \quad \text{s.t.} \quad Ax \leq b.$$

The Lagrangian is:

$$\mathcal{L}(x, \lambda) = c^\top x + \lambda^\top (Ax - b), \qquad \lambda \geq 0.$$

Dual ascent updates:

$$x^{t+1} = \arg\min_{x \in [0,1]^n} \left(c^\top x + \lambda^{t\top} Ax \right),$$

$$\lambda^{t+1} = [\lambda^t + \alpha(Ax^t - b)]_+.$$

A.2. MILP Attention as Dual Variables

Our MILP Attention layer computes:

$$H = A^\top v, \qquad v = \max(0, Ax - b).$$

In dual ascent:

$$\nabla_x \mathcal{L}(x, \lambda) = c + A^\top \lambda.$$

Thus:

$$\lambda \approx v, \quad \text{and} \quad A^\top v \approx A^\top \lambda,$$

making v an implicit dual estimate.

The attention step:

$$x \leftarrow \text{softmax}(-H) \odot x,$$

is equivalent to a dual-weighted projection that suppresses violations.

A.3. Constraint Experts as Nonlinear Dual Multipliers

Classical dual updates are linear in violations:

$$\lambda^{t+1} = [\lambda^t + \alpha v]_+.$$

Our MoE generalizes this:

$$\lambda^{(\text{learned})} = E_k(v),$$

allowing nonlinear penalty structure tailored to different constraint families.

A.4. Latent Gradient Refinement as Primal Update

Our model updates:

$$x \leftarrow \text{clip}(x - \eta \nabla_x \mathcal{L}(x)), \quad \mathcal{L}(x) = c^\top x + \Phi(x),$$

which corresponds to:

$$x^{t+1} = \Pi_{[0,1]^n} \left(x^t - \eta(c + A^\top \lambda) \right).$$

Thus the refinement loop is a differentiable surrogate of primal descent.

A.5. Summary

Dual Variables:	$v,\ A^\top v,\ E_k(v)$	$\approx$	λ
Dual Update:	$\lambda \leftarrow \lambda + (Ax - b)_+$	$\approx$	$\mathrm{MoE}(v)$
Primal Update:	$x \leftarrow x - \eta(c + A^\top \lambda)$	$\approx$	$x - \eta \nabla_x \mathcal{L}(x)$
Dual Projection:	$\exp(-(A^\top \lambda))$	$\approx$	$\mathrm{MILPAttn}(x)$

The neural solver therefore acts as an amortized, learned surrogate for primal-dual Lagrangian optimization.